Superpixels optimized by color and shape

Vitaliy Kurlin, Donald Harvey

Department of Computer Science, University of Liverpool, UK

Abstract. Image over-segmentation is formalized as the approximation problem when a large image is segmented into a small number of connected superpixels with best fitting colors. The approximation quality is measured by the energy whose main term is the sum of squared color deviations over all pixels and a regularizer encourages round shapes. The first novelty is the coarse initialization of a non-uniform superpixel mesh based on selecting most persistent edge segments. The second novelty is the scale-invariant regularizer based on the isoperimetric quotient. The third novelty is the improved coarse-to-fine optimization where local moves are organized according to their energy improvements. The algorithm beats the state-of-the-art on the objective reconstruction error and performs similarly to other superpixels on the benchmarks of BSD500.

Keywords: superpixel, segmentation, approximation, boundary recall, reconstruction error, energy minimization, coarse-to-fine optimization

1 Introduction: motivations, problem and contributions

1.1 Motivations: superpixels speed up higher level processing

Modern cameras produce images containing millions of pixels in a rectangular grid. This pixel grid is not the most natural nor most efficient representation, because not all these pixels are needed to correctly understand an image. Moreover, processing a large image pixel by pixel is slow, and many important algorithms have the running time $O(n^2)$ in the number n of pixels. However we know of a smart vision system (called a human brain) that quickly extracts key elements of complicated scenes by skipping the vast majority of incoming light signals.

The main challenge of low-level vision is to represent a large image in a less-redundant form that can speed up the higher level processing. The central problem is the unsupervised *over-segmentation* when a pixel-based image is segmented into *superpixels* (unions of square-based pixels), which are perceptually meaningful atomic regions with consistent features such as color or texture.

Our motivations are to address the following key challenges of superpixels:

• rigorously state the over-segmentation as an approximation problem when a large pixel-based image is approximated by a mesh of fewer superpixels;

• add constraints that superpixels are connected and have no inner holes;

• optimize superpixels in a data-driven way, e.g. by smartly choosing an original configuration and attempting steps in a good order according to their costs;

• avoid parameters whose influence on superpixels are hard to describe.

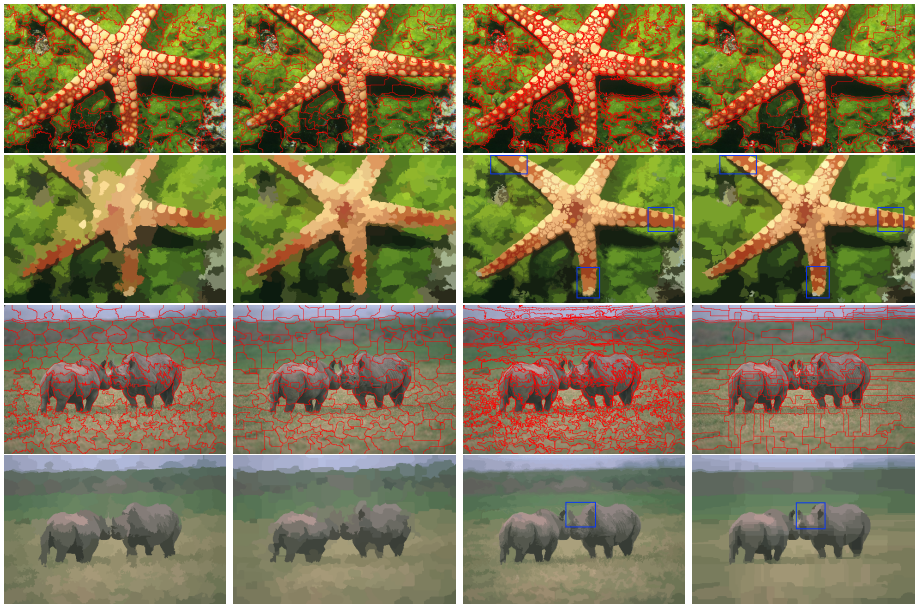

Fig. 1. Odd rows: superpixel meshes by algorithms SLIC [1], SEEDS [2], ETPS [3], ours. **Even rows**: Reconstructed images with the average color for every superpixel. Blue rectangles show the areas where our compact superpixels better capture details.

1.2 Oversegmentation by superpixels is an approximation problem

The aim is to segment an image of n pixels into at most $k < n$ *superpixels* that are connected unions of pixels satisfying conditions (1.2a)–(1.2d) below.

(1.2a) The resulting superpixels with best constant colors approximate the image well, e.g. a difference between an image and its approximation is minimized.

(1.2b) By construction the superpixels are connected and have no inner holes.

(1.2c) Superpixels adhere well to object boundaries, e.g. in comparison with human-drawn contours in the Berkeley Segmentation Database BSD500 [4].

(1.2d) The only parameters are the number of superpixels and a shape coefficient for a trade-off between the accuracy of boundaries and shapes of superpixels.

Since images are often replaced by their superpixel meshes, condition (1.2a) highlights the importance to measure the quality of such an approximation. The pixelwise sum of squared differences is the standard statistical mean error and can be based on colors (as in Definition 1) or on texture information or other pixel features. Condition (1.2b) guarantees that no post-processing is needed so that a superpixel mesh can be represented by a simple graph (instead of a much larger regular grid) whose nodes are superpixels and whose links connect adjacent superpixels. Condition (1.2c) follows the tradition to evaluate superpixels on BSD benchmarks. Condition (1.2d) restricts manually chosen parameters.

1.3 Contributions to the state-of-the-art for superpixels

First, we introduce an *adaptive initialization* of a superpixel mesh, whose main idea of persistent edges from Definition 3 can be used in any hot spot analysis. Second, the new regularizer in Definition 2 is scale-invariant, hence the superpixels are truly optimized by shape. Third, the optimization is improved in subsection 4.2 and its time is justified for the first time in Theorem 12. Here are the stages of the algorithm SOCS: Superpixels Optimized by Color and Shape.

Stage 1: detecting persistent horizontal and vertical edges along object boundaries to form a non-uniform grid of rectangular blocks, see subsection 3.2.

Stage 2: merging blocks in a grid when a reconstruction error is minimally increased to get a non-regular initial mesh that is quickly adapted to a given image and contains a required number of superpixels, see subsection 3.4.

Stage 3: subdividing rectangular blocks within every superpixel into sub-blocks going from a coarse level to a finer level of optimization in subsection 4.1.

Stage 4: a new way to choose boundary blocks for moving to adjacent superpixels, then repeat Stage 3 until all blocks become pixels, see subsection 4.2.

2 A review of past superpixel algorithms

The excellent survey by D. Stutz et al. [5, table 3 in section 8] recommends 6 algorithms, which are reviewed below in addition to few other good methods.

A pixel-based image is represented by a graph G whose nodes are in a 1–1 correspondence with all pixels, while all edges of G represent adjacency relations between pixels, when each pixel is connected to its closest 4 or 8 neighbors.

The seminal *Normalized Cuts* algorithm by Shi and Malik [6] finds an optimal partition of G into connected components, which minimizes an energy taking into account all nodes of G. The Entropy Rate Superpixels (ERS) of Lie et al. [7] minimizes the entropy rate of a random walk on a graph. Based on *Compact Superpixels* by Veksler and Boykov [8], the faster algorithm by Zhang et al. [9] processes an average image from BSD500 in 0.5 sec. The Contour Relaxed Superpixels (CRS) by Conrad et al. [10] optimize a cost depending on texture.

The *Simple Linear Iterative Clustering* (SLIC) algorithm by Achanta et al. [1] forms superpixels by k-means clustering in a 5-dimensional space using 3 colors in CIELAB space and 2 coordinates per pixel. Because the search is restricted to a neighborhood of a given size, the complexity is $O(kmn)$, where n is the number of pixels and m is the number of iterations. This gives an average running time of about 0.2s per image in BSD500. If a final cluster of pixels is disconnected or contains holes, post-processing is possible, but increases the runtime.

The recent improvements of SLIC are the *Linear Spectral Clustering* (LSC) by Li et al. [11] based on a weighted k-means clustering in a 10-dimensional space, and the Eikonal Region Growing Clustering (EGRC) by Buyssens et al.[12].

The coarse-to-fine optimisation progressively approximates a superpixel segmentation. At the initial coarse level, each superpixel consists of large rectangular

blocks of pixels. At the next level, all blocks are subdivided into 4 rectangles and one rearranges the blocks to find a better approximation depending on a cost function, which continues until all blocks become pixels.

SEEDS (Superpixels Extracted via Energy-Driven Sampling) by Van den Bergh et al. [2] seems the first superpixel algorithm to use a coarse-to-fine optimization. The colors of all pixels within each fixed superpixel are put in bins, usually 5 bins for each channel. Each superpixel has the associated sum of deviations of all bins from an average bin within the superpixel. This sum is maximal for a superpixel whose pixels have colors in one bin. SEEDS iteratively maximizes the sum of deviations by shrinking or expanding superpixels.

The ETPS algorithm (Extended Topology Preserving Superpixels) by Yao et al. [3] minimizes a different cost function, which is the reconstruction error RE in subsection 3.1 plus the deviation of pixels within a superpixel from a geometric center, along with a cost proportional to the boundaries of superpixels. This regularizer encourages superpixels of small sizes, however the benchmarks on BSD500 are computed [3, Fig. 4] without the regularizer (as for SEEDS).

SEEDS and ETPS satisfy topological Condition (1.2c) by construction. ETPS was highlighted as the best algorithm by D. Stutz et al. [5, table 3 in section 8].

3 Energy-based superpixels formed by coarse blocks

This section explains the new adaptive initialization for coarse superpixels that are better than a uniform grid, which is used in most past superpixel algorithms. Persistent edges in a given image generate a non-uniform mesh of rectangular blocks. These blocks are iteratively merged in such a way that the energy function remains as small as possible or until we get a maximum number of superpixels.

3.1 The energy is a reconstruction error of approximation

An image I can be considered as a function from pixels to a space of colors. We consider $I(p)$ as the vector (L, a, b) of 3 colors in the CIELAB space, which is more perceptually uniform than *RGB space* with red, green, blue components.

In the *CIELAB space*, L is the lightness, a represents the colors from red to green (the lowest value of a means red, the highest value of a means green). The component b similarly represents the opponent colors from yellow to blue. The OpenCV function cvtColor outputs each Lab channel in the range $[0, 255]$.

Definition 1 *Let an image I of n pixels be segmented into k superpixels. For every pixel p, denote by $S(p)$ the superpixel containing p. Then $S(p)$ has the mean* $\operatorname{color}(S(p)) = \dfrac{1}{|S(p)|} \displaystyle\sum_{q \in S(p)} I(q)$. *Since I is approximated by superpixels with mean colors, the natural measure of quality is the* Reconstruction Error

$$RE = \textit{sum of squared color deviations} = \sum_{p=1}^{n} \Big(I(p) - \operatorname{color}(S(p)) \Big)^2. \qquad (3.1a)$$

Each of the 3 colors in the Lab space has the range $[0, 255]$. *Hence the following normalized Root Mean Square of the color error in percents is shown in Fig. 6.*

$$RMS = \sqrt{\frac{RE}{3n}} \times \frac{100\%}{255} = \sqrt{\frac{1}{3n} \sum_{p=1}^{n} \Big(I(p) - \text{color}(S(p))\Big)^2} \times \frac{100\%}{255}. \quad (3.1b)$$

The Reconstruction Error RE can be written similarly to (3.1a) for other pixel properties instead of colors, e.g. for texture. The main objective advantage of $nRMS$ in (3.1a) is its independence of any subjective ground-truth.

A color term proportional to RE was used by Yao et al. [3] with the regularizer PD = sum of squared pixel deviations = $\sum_{p} \Big(p - \text{center}(S(p))\Big)^2$, where $\text{center}(S(p)) = \frac{1}{|S(p)|} \sum_{q \in S(p)} q$ is the geometric center of the superpixel $S(p)$. The above term PD is not invariant under scaling and penalizes large superpixels, which has motivated us to introduce the scale-invariant regularizer.

Definition 2 *The* isoperimetric quotient $\text{IQ}(S) = \dfrac{4\pi \text{ area}(S)}{\text{perimeter}^2(S)}$ *of a superpixel* S *is a scale-invariant shape characteristic having the maximum value 1 for a round disk* S. *The* IQ measure *of an image over-segmentation* $I = \cup S_{i=1}^{k}$ *is*

$$\textit{the average IQ} = \sum_{\textit{superpixels } S} \frac{\text{IQ}(S)}{\#\textit{superpixels}} = \frac{4\pi}{k} \sum_{i=1}^{k} \frac{\text{area}(S_i)}{\text{perimeter}^2(S_i)}. \quad (3.1c)$$

The SOCS algorithm will minimize the energy equal to the weighted sum

$$\textit{Energy} = \frac{RE}{n} + c \times \text{IQ}, \ \textit{where RE is in (3.1a), c is a shape coefficient.} \ (3.1d)$$

Schick et al. [13] suggested another weighted average of isoperimetric quotients $\text{CO} = \sum_{\text{superpixels S}} \frac{\text{area}(S)}{\#\text{superpixels}} \text{IQ}(S) = \frac{4\pi}{k} \sum_{i=1}^{k} \frac{\text{area}^2(S_i)}{\text{perimeter}^2(S_i)}$, when larger superpixels are forced to have more round shapes, see experiments Fig. 6.

3.2 Stage 1: detection of persistent horizontal and vertical edges

The SOCS algorithm starts by finding *persistent edges* along horizontal and vertical lines of a pixel grid, see Definition 3. The first step is to apply to a given image I the bilateral filter from OpenCV with the size of 5 pixels and sigma values 100 for deviations in the color and coordinate spaces. The second step is to compute the image gradients $d_x I$ and $d_y I$ using the standard 2×2 masks. For every row $j = 1, \ldots,$rows(I) in a given image I, we have a graph of gradients $|d_y(I)|$ over $1 \le i \le$ columns(I). The similar graph of magnitudes $|d_x(I)|$ can be computed over every column of I. For any such graph of discrete values $f(1), \ldots, f(l)$, Definition 3 formalizes the automatic method to detect continuous intervals $a \le t \le b$, where the graph f has persistently high values.

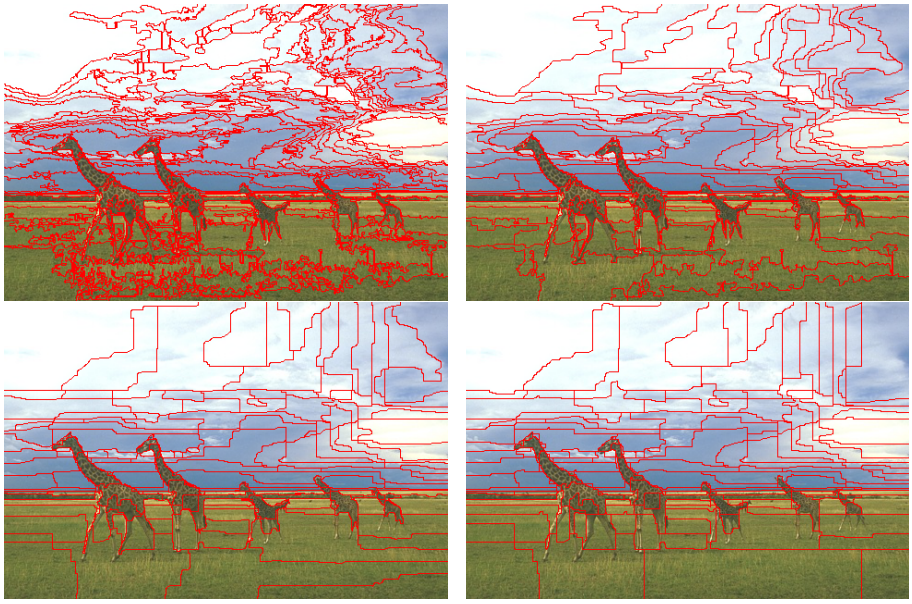

Fig. 2. The effect of c in (3.1d): **1st**: $c = 0$, **2nd**: $c = 1$, **3rd**: $c = 10$, **4th**: $c = 100$.

Definition 3 *For a function $f : \mathbb{R} \to \mathbb{R}$ discretely sampled at $t = 1, \dots, l$, the* strength *of a line edge $L = [a, b]$ is the sum $\sum_{t \in [a,b]} f(t)$. Fig. 3 visualises the strength of an edge L as the area under a continuous graph $f(t)$ over L. For any threshold v, the* superlevel *set $f^{-1}[v, +\infty) = \{t \in \mathbb{R} : f(t) \geq v\}$ consists of several edges L_i. When v is decreasing, the edges L_i are growing and merge with each other until we get a single edge covering all points $t = 1, \dots, l$. For any fixed v, we compute the widest gap between the strengths of the edges that form the superlevel set $f^{-1}[v, +\infty)$. We find a critical level v between the median and maximum of the widest gap above is maximal, see Fig. 3. At this critical value v, the edges whose strengths are above the widest gap are called* persistent.

Proofs of claims can be replaced by more image experiments in a final version.

Lemma 4 *For any image I of a size $w \times h$ pixels, the persistent edges in all $w+h$ horizontal and vertical lines in I can be found in time $O(w \log h + h \log w)$.*

Proof. Assuming that the points $t = 1, \dots, l$ form a connected interval graph, the segments in Definition 3 are connected components of a superlevel set $f^{-1}[v, +\infty)$. These components are maintained by a union-find structure, which requires $O(\log l)$ operations per update (creating a new segment, adding a new node to an old segment or merging 2 segments). Every update requires changes of strengths for at most 2 segments, hence $O(\log l)$ operations we if keep the ordered set of strengths in a binary tree. The time is $O(w \log h + h \log w)$ for w columns (vertical lines) of length h and h rows (horizontal lines) of length w. □

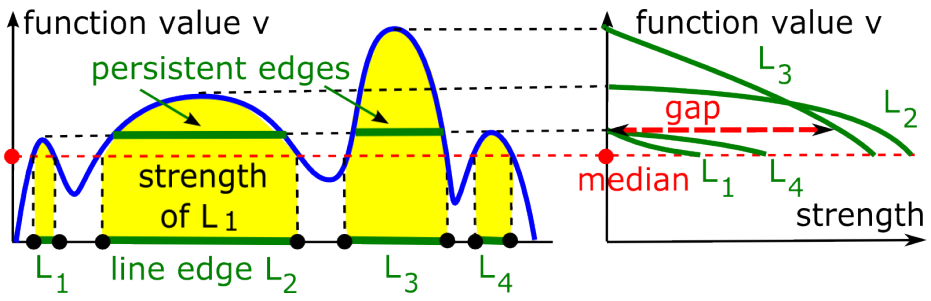

Fig. 3. Left: a superlevel set has 4 edges in green with their strengths highlighted as yellow areas at the median value of f. **Right**: strengths of edges are analyzed when a threshold v is decreasing, the widest gap between strengths of edges is shown in red.

Given an expected number k of superpixels, the average area of a single superpixel is n/k. If such a superpixel is a square, its side would be $s = \sqrt{n/k}$. If an image I has a size $w \times h$, we build the a non-uniform grid of $2[w/s] \times 2[h/s]$ rectangular blocks. We select $2[w/s]$ columns and $2[h/s]$ rows that have the maximum strengths of their persistent edges from Definition 3. To avoid close edges, after selecting a current maximum along a line $x =$const or $y =$const, we later ignore the neighboring lines at a distance less than 4 pixels. By extending persistent edges to the boundary of I, we get a non-uniform *edge grid*, see Fig. 4.

3.3 Cost of merging superpixels and the superpixel structure

The edge grid is already adapted to the image I better than the standard uniform grid used in other algorithms. However, large regions of almost constant colors such as sky can be cut by extended edges into unnecessary small blocks. Stage 2 in subsection 3.4 will merge rectangular blocks into a smaller number of larger superpixels without increasing the Reconstruction Error too much.

If an image is segmented into superpixels $I = \cup_{i=1}^{k} S_i$, the Reconstruction Error in formula (3.1a) decomposes as a sum of energies over all superpixels:

$$RE = \sum_{i=1}^{k} E(S_i), \text{ where } E(S_i) = \sum_{p \in S_i} \Big(I(p) - \text{color}(S_i) \Big)^2. \qquad (3.3a)$$

$$\text{The cost of merging } S_i, S_j \text{ is } E(S_i, S_j) = E(S_i \cup S_j) - E(S_i) - E(S_j) \geq 0. \quad (3.3b)$$

This cost can be 0 only if S_i, S_j have exactly the same mean $\text{color}(S_i) = \text{color}(S_j)$. Technically, two superpixels S_i, S_j may share more than one edge, e.g. a connected chain of edges. If the intersection $S_i \cap S_j$ is disconnected, e.g. one edge e and a vertex $v \notin e$, we set $E(S_i, S_j) = +\infty$, so the superpixels S_i, S_j will not merge to avoid harder cases when a superpixel may touch itself.

To prepare the coarse-to-fine optimization in section 4 when superpixels are iteratively improved, we introduce the superpixel structure from our implementation with sums and pointers to 4 sub-blocks for each rectangular block.

Fig. 4. Left: red persistent edges generate the blue edge grid at Stage 1. **Middle**: initial mesh after Stage 2. **Right**: final mesh with 99 superpixels after Stages 3-4.

We split any rectangular block from the non-uniform grid into the four smaller rectangular *sub-blocks* by subdividing each side into 2 almost equal parts whose lengths differ by at most 1 pixel. We don't subdivide 1-pixel sides, so 1-pixel wide blocks are subdivided into 2 blocks. Since each block may keep pointers to its 4 sub-blocks, the superpixel structure looks like a large tree where the root points to coarsest blocks each of which points to its 4 sub-blocks and so on.

Definition 5 *The* superpixel structure *of S contains the number $|S|$ of pixels in a superpixel S,* $\mathrm{sum}(S) = \sum_{p\in S} I(p)$, $\mathrm{sum2}(S) = \sum_{p\in S} \Big(I(p)\Big)^2$, *the list of (x, y) indices in the block grid of blocks covered by S. Each block B has the index of its superpixel S, similar sums $|B|, \mathrm{sum}(B), \mathrm{sum2}(B)$ and pointers to its 4 sub-blocks.*

The color sums of a superpixel S in Definition 5 are justified by Lemma 6.

Lemma 6 *In (3.3b) the cost $S(E_i, E_j)$ of merging superpixels S_i, S_j can be computed by using the structure of S_i, S_j from Definition 5 in a constant time.*

Proof. Since $\mathrm{sum}(S) = |S|\mathrm{color}(S)$, the energy in (3.3b) becomes

$$E(S) = \sum_{p\in S} \left(\Big(I(p)\Big)^2 - 2\mathrm{color}(S)I(p) + \mathrm{color}(S)^2 \right) = \sum_{p\in S} \Big(I(p)\Big)^2 -$$

$$-2\mathrm{color}(S) \sum_{p\in S} I(p) + |S| \Big(\mathrm{color}(S)\Big)^2 = \mathrm{sum2}(S) - \frac{\Big(\mathrm{sum}(S)\Big)^2}{|S|}. \qquad (3.3c)$$

Since the union $S_i \cup S_j$ nicely affects the area and sums, i.e. $|S_i \cup S_j| = |S_i| + |S_j|$,

$$\mathrm{sum}(S_i \cup S_j) = \mathrm{sum}(S_i) + \mathrm{sum}(S_j), \quad \mathrm{sum2}(S_i \cup S_j) = \mathrm{sum2}(S_i) + \mathrm{sum2}(S_j),$$

(3.3c) implies that the computation of $E(S_i, S_j)$ is independent of $|S_i \cup S_j|$. □

3.4 Stage 2: merging adjacent superpixels with minimum energy

At Stage 2 adjacent superpixels are iteratively merged starting from pairs with a minimum cost $E(S_i, S_j)$ in (3.3b). Since superpixels may share more than

one edge, we associate the cost $E(S_i, S_j)$ to pairs of adjacent superpixels. Each unordered pair (S_i, S_j) has a unique key(S_i, S_j), e.g. formed by indices of superpixels in the edge grid. Stage 2 finishes when the number of superpixels drops down to a given maximum m. Fig. 6 shows experiments where the number of superpixels can go down to $0.25m$ if $nRMS$ jumps by not more than 2%.

Lemma 7 *If an image $I = \cup_{i=1}^{k} S_i$ is segmented into k superpixels, there are at most $O(k)$ pairs (S_i, S_j) of adjacent superpixels. In time $O(k \log k)$ one can find and merge (S_i, S_j) with a minimal cost $E(S_i, S_j)$ updating the costs of all pairs.*

Proof. Since the common boundary of S_i, S_j grows over time, we keep the list of all common edges in the binary *edge tree* indexed by key(S_i, S_j), which allows a fast insertion and deletion of new pairs of adjacent superpixels. To quickly find key(S_i, S_j) and the corresponding pair of adjacent superpixels with a minimum cost $E(S_i, S_j)$, we put all keys into the binary *cost tree* indexed by $E(S_i, S_j)$.

All k superpixels form a planar network with f bounded faces, g edges, where each pair of adjacent superpixels is represented by one edge. Since each face f has at least $3f$ edges, the doubled number of edges $2g$ is at least $3f$, so $f \le \frac{2}{3}g$. The Euler formula $k - g + f = 1$ gives $1 \le k - g + \frac{2}{3}g$, hence $g \le 3(k-1)$.

Then both binary trees above have the size $O(k)$. The first element in the cost tree has the minimum cost $E(S_i, S_j)$ and can be found and removed in a constant time. The search for the corresponding key(S_i, S_j) in the edge tree in time $O(\log k)$ leads to the list of common edges of the superpixels S_i, S_j.

The edge grid from is converted into a polygonal mesh using the OpenMesh library. Then each common edge is removed by the collapse and remove_edge operations from OpenMesh taking a constant time. For each of remaining $O(k)$ edges of $S_i \cup S_j$ on the boundary of another superpixel S, the cost $E(S, S_i \cup S_j)$ is computed by Lemma 6 and is added to the cost tree in time $O(\log k)$. □

4 The coarse-to-fine optimization for superpixels

This section carefully analyzes the coarse-to-fine optimization used by Yao et al. [3]. At Stage 3 each rectangular block in a current grid is subdivided as explained before Definition 5. At Stage 4 each *boundary block* that belongs to a superpixel S_i and is adjacent to another superpixel S_j is checked for a potential move from S_i to S_j. After completing this optimization for all boundary blocks, Stages 3 and 4 are repeated at the next finer level until all blocks become pixels.

4.1 Stage 3: subdividing rectangular blocks into four sub-blocks

Lemma 8 explains how the superpixel structure from Definition 5 helps us to quickly compute color sums for all superpixels and subdivide all superpixels.

Lemma 8 *Let an image $I = \cup_{i=1}^{k} S_i$ of n pixels be segmented into k superpixels. Then all $|S_i|, \mathrm{sum}(S_i), \mathrm{sum2}(S_i)$ can be found in time $O(n)$ independent of k.*

Proof. We recursively compute all sums for each block B by adding the corresponding sums from each of 4 sub-blocks of B. Since, for each single-pixel block $B = p$, we have $|B| = 1$, $\mathrm{sum}(B) = I(p)$, $\mathrm{sum2}(B) = \Big(I(p)\Big)^2$, we need only $O(n) + O(n/4) + O(n/16) + \cdots = O(n)$ additions to compute the sums for all blocks. For each superpixel S_i, we find $|S_i|, \mathrm{sum}(S_i), \mathrm{sum2}(S_i)$ by adding the sums from all blocks in S_i in time $O(|S_i|)$, so the total time is $O(n)$. □

Lemma 9 *When blocks are subdivided going from a coarse to a finer level, each superpixel S containing b blocks larger than 1×1 can be updated in time $O(b)$.*

Proof. By Definition 5 for each superpixel S, we only need to replace the list of blocks in the current grid by a longer list of blocks in the refined grid, which is done by merging the lists from the 4 sub-blocks of each block covering by S. The index of S is copied to every new sub-block, which takes $O(b)$ time. □

4.2 Tree of boundary blocks and local connectivity of superpixels

A block B in a superpixel S is called *boundary* if one of its 4 side neighbors belongs to a different superpixel, which can be quickly checked by comparing superpixel indices of all blocks. The ETPS algorithm puts all boundary blocks into a *priority queue* and adds any new boundary blocks to the end of this queue. We have replaced this queue by a binary tree where blocks are ordered by costs of moves so that moves are attempted according to their costs, not row by row.

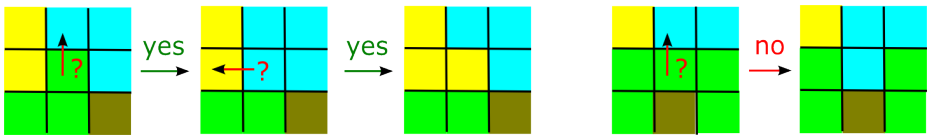

Fig. 5. Left: allowed moves preserves the local connectivity. **Right**: a forbidden move.

Blocks in the tree are tested one by one for a potential move to an adjacent superpixel. Such a move was called *forbidden* in [3, section 3] if S becomes disconnected after removing B. However, the global connectivity of $S - B$ is slow to check. A removal of a boundary block B from a superpixel S respects the *local connectivity* of S if the 8-neighborhood of B within $S - B$ is connected, see Fig. 5. The 3 pictures in [3, Fig. 3] show some (but not all) forbidden moves, so we justify below why the local connectivity can be checked in a constant time.

Lemma 10 *For any boundary block B moving from a superpixel S_i to another superpixel S_j, the* local connectivity *of $S_i - B$ can be checked in a constant time.*

Proof. We go around the circular 8-neighborhood $N_8(B)$, consider all blocks of $S-B$ as isolated vertices, add an edge between vertices u, v if the corresponding blocks in $N_8(B)$ share a common side. Then $S-B$ is locally connected around B if and only if the resulting graph on at most 8 vertices is connected. □

4.3 Stage 4: updating superpixels in a constant time per move

Let the move of $B \subset S_i$ to another superpixel S_j keep $S_i - B$ locally connected. If the Reconstruction Error in (3.1a) is decreased, we move B to S_j and will add new boundary blocks to the cost tree, otherwise we remove B from the tree.

Lemma 11 *For any block B moving from a superpixel S_i to another superpixel S_j, the structures of both superpixels S_i, S_j can be updated in a constant time.*

Proof. All sums of colors over the block B are subtracted from the corresponding sums of S_i and are added to the sums of S_j. We change the superpixel index of B from i to j. After B has moved, only its 4 neighboring blocks can change their boundary status, which is checked in a constant time by comparing superpixel indices. Any new boundary blocks are added to the binary tree of blocks. □

The time at Stage 4 essentially depends on the number q of boundary blocks that are processed in the cost tree. Stage 4 finishes when the tree is empty or q exceeds the upper bound of n given pixels, which never happened for BSD500.

Theorem 12 *The SOCS algorithm segmenting an image of n pixels into k superpixels has the asymptotic computational complexity $O(n + k^2 \log k + q)$.*

Proof. Stage 1 has time about $O(\sqrt{n} \log n)$ by Lemma 4. At Stage 2 we merge at most $O(k)$ pairs of superpixels, each pair in time $O(k \log k)$ by Lemma 7. By Lemma 8 all superpixels are subdivided in time $O(b)$ for a grid with b blocks larger than 1×1. The number of such blocks increases to n by a factor of at least 2, hence the total time for Stages 3 is $O(n)$. By Lemmas 10 and 11 the time for Stage 4 is proportional to the number q of processed blocks in the cost tree, because each boundary block is adjacent to at most 3 other superpixels. □

5 Comparisons with other algorithms on BSD500

The Berkeley Segmentation Database BSD500 [4] contains 500 natural images and human-drawn closed contours around object boundaries. Then all pixels in every image are split into disjoint *segments*, which are large unions of pixels comprising a single object. So every pixel has the index of its superpixel and some pixels are also labeled as boundary. Every image has about 5 human drawings, which vary significantly and are called the *ground-truth segmentations*.

For an image I, let $I = \cup G_j$ be a segmentation into ground-truth segments and $I = \cup_{i=1}^{k} S_i$ be an oversegmentation into superpixels produced by an algorithm. Each quality measure below compares the superpixel $S_1, \dots, S_k$ with the best suitable ground-truth from the BSD500 database for every image.

Let $G(I) = \cup G_j$ be the union of ground-truth boundary pixels and $B(I)$ be the boundary pixels produced by a superpixel algorithm. For a distance ε in pixels, the Boundary Recall $BR(\varepsilon)$ is the ratio of ground-truth boundary pixels $p \in G(I)$ that are within the distance ε from the superpixel boundary $B(I)$.

$$\text{The Undersegmentation Error } UE = \frac{1}{n} \sum_j \sum_{S_i \cap G_j \neq \emptyset} |S_i - G_j| \qquad (5a)$$

was often used in the past, where $|S_i - G_j|$ is the number of pixels that are in S_i, but not in G_j. However a superpixel is fully penalized when $S_i \cap G_j$ is 1 pixel, which required ad hoc thresholds, e.g. the 5% threshold $|S_i - G_j| \geq 0.05|S_i|$ by Achanta et al. [1], or ignoring boundary pixels of S_i by Liu et al. [7].

Van den Bergh et al. [2] suggested the more accurate measure, namely

$$\text{the Corrected Undersegmentation Error } CUE = \frac{1}{n} \sum_i |S_i - G_{max}(S_i)| \quad (5b),$$

where $G_{max}(S_i)$ is the ground-truth segment having the largest overlap with S_i. Neubert and Protzel [14] introduced the Undersegmentation Symmetric Error

$$USE = \frac{1}{n} \sum_j \sum_{S_i \cap G_j \neq \emptyset} \min\{\, in(S_i),\, out(S_i) \,\}, \text{ where} \qquad (5c),$$

$in(S_i)$ is the area of S_i inside G_j, $out(S_i)$ is the area of S_i outside G_j. To keep graphs readable, Fig. 6 compares SOCS to the 3 past algorithms ETPS, SEEDS, SLIC, coming on top of others in the evaluations by Stutz et al. [5, Table 3].

As suggested by Theorem 12, the running time of SOCS is similar to ETPS at about 1s on average per BSD image on a laptop with 2.6 GHz and 8G RAM.

6 Summary and discussion of the new SOCS algorithm

The SOCS algorithm has a fast adaptive initialization that is based on persistent edges in an image and can substantially reduce the number of superpixels without compromizing the quality of approximation. The new coarse-to-fine optimization quickly converges to a minimum by moving boundary blocks of large sizes and then by subdividing them into smaller blocks many of which remain stable.

- The first theoretical contribution is the formal statement of the image oversegmentation as an approximation problem by superpixels in subsection 1.2.
- The adaptive initialization of superpixels consisting of large rectangular blocks can be used in many other algorithms that start from a coarse uniform grid.
- The coarse-to-fine optimization has been substantially improved by keeping boundary blocks sorted in a binary tree instead of a linear queue.
- The SOCS algorithm outperforms the state-of-the art for the approximation error (nRMS) and undersegmentation errors CUE/USE on BSD500 images.

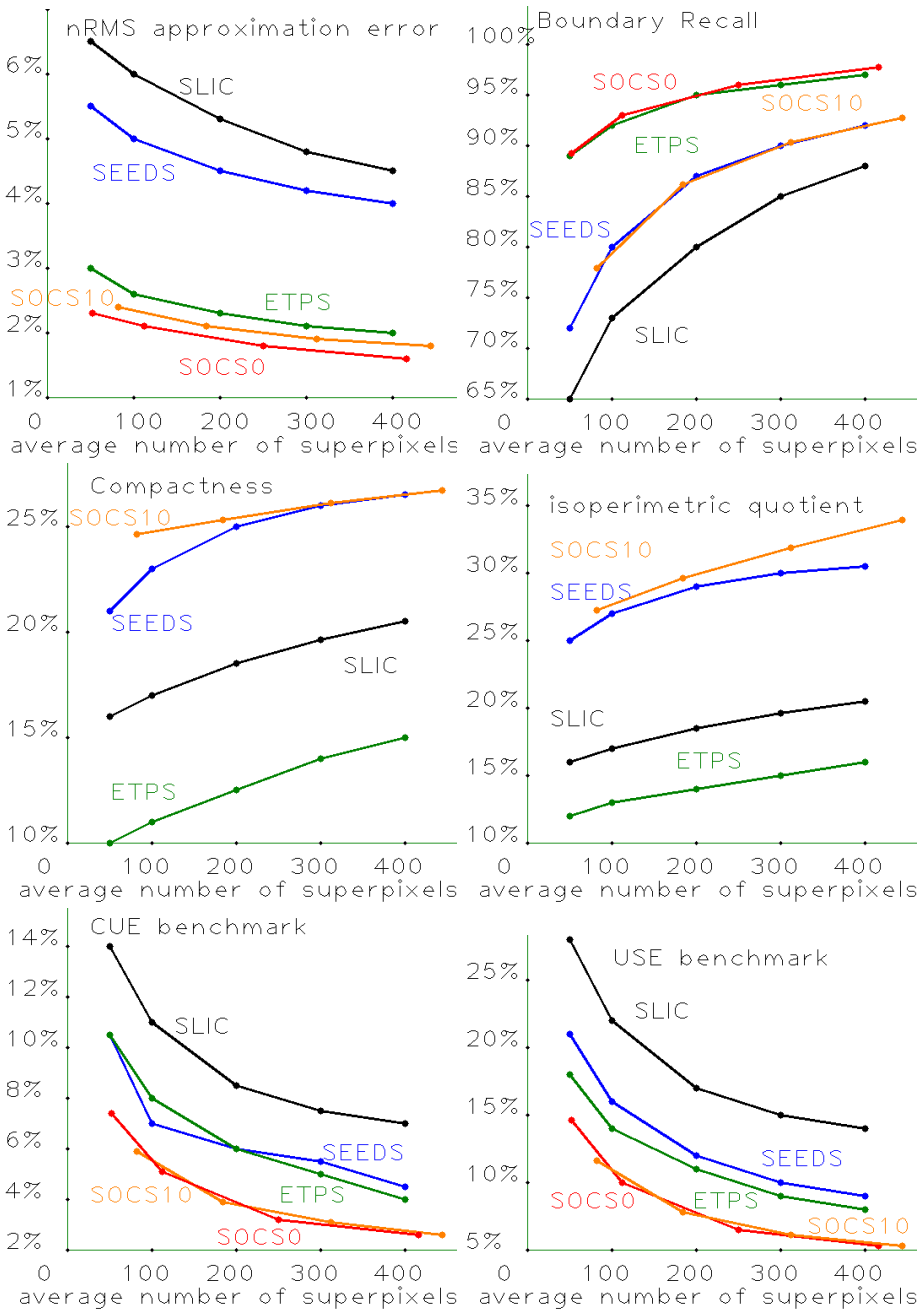

Fig. 6. Each dot is (average number of superpixels, average benchmark) on BSD500. SOCS0 in red has shape coefficient $c = 0$, SOCS10 in orange has $c = 10$, see (3.1d).

Here are the practical advantages of the new SOCS algorithm.

• The output superpixels are *connected*, because the connectivity is checked in Stage 4 when boundary blocks are updated, which gives the overall speed-up.

• The SOCS algorithm can be stopped at any time after Stage 1, e.g. at any optimization step, because each update needs a constant time by Lemma 11.

• The only *essential input parameters* are the maximum number k of superpixels and the shape coefficient for a trade-off between accuracy and compactness.

• The SOCS algorithm is *modular* and allows improvements in different parts, e.g. persistent edges in Stage 1 can be found in another way, merging blocks in Stage 2 can be done by another strategy with the same Reconstruction Error.

We are happy to publish the C++ code based on OpenCV and OpenMesh in September 2017, and thank all anonymous reviewers for their helpful suggestions.

References

1. Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., Süsstrunk, S.: Slic superpixels compared to the state-of-the-art. Transactions PAMI **34** (2012) 2274–2282
2. Van de Bergh, M., Boix, X., Roig, G., Van Gool, L.: Seeds: superpixels extracted via energy-driven sampling. Int J Computer Vision **111** (2015) 298–314
3. Yao, J., Boben, M., Fidler, S., Urtasun, R.: Real-time coarse-to-fine topologically preserving segmentation. In: Proceedings of CVPR. (2015) 216–225
4. Arbelaez, P., Maire, M., Fowlkes, C., Malik, J.: Contour detection and hierarchical image segmenetaton. Transactions PAMI **33** (2011) 898–916
5. Stutz, D., Hermans, A., Leibe, B.: Superpixels: An evaluation of the state-of-the-art. Computer Vision and Image Understanding (2017)
6. Shi, J., Malik, J.: Normalized cuts and image segmentation. Transactions PAMI **22** (2000) 888–905
7. Liu, M.Y., Tuzel, O., Ramalingam, S., Chellappa, R.: Entropy rate superpixel segmentation. In: Proceedings of CVPR. (2011) 2097 – 2104
8. Veksler, O., Boykov, Y., Mehrani, P.: Superpixels and supervoxels in an energy optimization framework. In: Proceedings of ECCV. (2010) 211–224
9. Zhang, Y., Hartley, R., Mashford, J., Burn, S.: Superpixels via pseudo-boolean optimization. In: Proceedings of ICCV. (2011) 211–224
10. Conrad, C., Mertz, M., Mester, R.: Contour-relaxed superpixels. In: Proc. Energy Minimization Methods in Computer Vision & Pattern Recognition. (2013) 280–293
11. Li, Z., Chen, J.: Superpixel segmentation using linear spectral clustering. In: Proceedings of CVPR. (2015) 1356–1363
12. Buyssens, P., Toutain, M., Elmoataz, A., Lézoray, O.: Eikonal-based vertices growing and iterative seeding for efficient graph-based segmentation. In: Int. Conf. Image Processing (ICIP). (2014) 4368–4372
13. Schick, A., Fischer, M., Stifelhagen, R.: Measuring and evaluating the compactness of superpixels. In: Proceedings of ICPR. (2012) 930–934
14. Neubert, P., Protzel, P.: Compact watershed and preemptive slic: On improving trade-offs of superpixel segmentation algorithms. In: Proc. ICPR. (2014) 996–1001